

Appendix: Logical Specifications-guided Dynamic Task Sampling for Reinforcement Learning Agents

Anonymous submission

A $LSTS^{ct}$ - Detailed Algorithm

A detailed algorithm for $LSTS^{ct}$ is described in Algo. 1. While the overall algorithm is similar to $LSTS$ (described in Section 4), $LSTS^{ct}$ differs from $LSTS$ in lines 9-29. In $LSTS$, while learning a policy for the task $\text{Task}(q, p)$, which corresponds to the transition $q \xrightarrow{\phi(q,p)} p$, we reinitialize the environment to a state $s \in \mathcal{S}_0$ once the agent reaches a state where the propositions for p hold true.

Instead of reinitializing the environment after reaching such a state where the propositions for p hold true, we let the Teacher agent sample a task (let's say $\text{Task}(p, r)$) from the set $\mathcal{X}[p] \setminus \text{DT}$, where \mathcal{X} is the adjacency matrix for the graph, and DT is the set of Discarded Tasks, as defined in Algo. 1. This helps the agent continue its training by attempting to learn a policy $\pi_{(p,r)}$ for $\text{Task}(p, r)$ while simultaneously learning a separate policy $\pi_{(q,p)}$ for the task $\text{Task}(q, p)$. If the agent fails to satisfy $\text{Task}(p, r)$, we reinitialize the environment from $s \sim \mathcal{S}_0$. Otherwise, the agent continues its training until it satisfies the high-level objective ϕ .

B Convergence criteria for a trajectory to satisfy the high-level objective

Given the ordered list of policies Π_{list}^* , we can generate a trajectory ζ in the task M with $\Pr_{\zeta \in \mathcal{Z}}[\zeta \text{ satisfies } \phi] \geq \eta$

A trajectory ζ is defined as a sequence of state sequences (s_0, s_1, \dots) where the transitions are defined by the MDP transition function $P : p(s'|s, a)$.

We define that a trajectory ζ satisfies the high-level objective ϕ if the initial state of agent in the trajectory s_0 satisfies the propositions corresponding to the node q_0 of the DAG \mathcal{G}_ϕ that represents ϕ , and the agent reaches a state s_f that satisfies the propositions corresponding to a node that lies in the set of accepting nodes $q_f \in F$ of the DAG \mathcal{G}_ϕ .

$\Pr_{\zeta \in \mathcal{Z}}[\zeta \text{ satisfies } \phi]$ denotes the probability with which a trajectory $\zeta \in \mathcal{Z}$ will satisfy ϕ . The set of all trajectories \mathcal{Z} are generated by the agent starting in a state $s \sim \mathcal{S}_0$ and following trajectories given by the policies in Π_{list}^* .

Since $\Pi_{list}^* = [\pi_{(q_1, q_2)}, \pi_{(q_2, q_3)}, \dots, \pi_{(q_{f-1}, q_f)}]$, and the list of MDP states visited by the trajectory ζ is given by $\zeta_s : (s_0, s_1, \dots)$.

The trajectory ζ satisfies the DAG \mathcal{G}_ϕ if for the sequence of states in the trajectory ζ_s :

$(\zeta_s[0], \dots, \zeta_s[k_1], \dots, \zeta_s[k_2], \dots, \dots, \zeta_s[k_f])$:

- $\zeta_s[0] \sim \mathcal{S}_0$
- $\zeta_0 \xrightarrow{L(\zeta_s[k_1])} \zeta_1 \xrightarrow{L(\zeta_s[k_2])} \zeta_2, \dots, \zeta_{f-1} \xrightarrow{L(\zeta_s[k_f])} \zeta_f$
- $L(\zeta_s[k_f]) \in F$

Thus, ζ_s contains states $\zeta_s[k_i]$ that triggers transitions of the agent's high-level state in the DAG \mathcal{G}_ϕ from the node q_i to the node q_j . Additionally, the initial state of the agent $\zeta_s[0]$ lies in the set of starting states \mathcal{S}_0 of the MDP M corresponding to the node q_0 and final state of the agent $\zeta_s[k_f]$ lies in the set of final states \mathcal{S}_f of the MDP M corresponding to the node $q_f \in F$.

As defined in Section 4, the *convergence criteria* for a task $\text{Task}(q_i, q_{i+1})$ ensures that the successful policy $\pi_{(q_i, q_{i+1})} \in \Pi_{list}^*$ achieves $\Pr_{\zeta \in \mathcal{Z}}[\zeta \text{ satisfies } \text{Task}(q_i, q_{i+1})] \geq \eta$ for all $i \in 1, \dots, f-1$.

Thus, convergence criteria for $\pi_{(q_0, q_1)}$ ensures that:

$$\Pr_{\zeta \in \mathcal{Z}}[\zeta_s[0 : i] \text{ satisfies } \text{Task}(q_0, q_1)] \geq \eta$$

Next, the convergence criteria for $\pi_{(q_1, q_2)}$ ensures that:

$$\Pr_{\zeta \in \mathcal{Z}}[\zeta_s[0 : i] \text{ satisfies } \text{Task}(q_0, q_1) \wedge \zeta_s[i : i+j] \text{ satisfies } \text{Task}(q_1, q_2)] \geq \eta \quad (1)$$

Following this procedure inductively, the convergence criteria for $\pi_{q_{f-1}, \sigma_{f-1}}$ ensures that:

$$\Pr_{\zeta \in \mathcal{Z}}[\zeta_s[0 : i] \text{ satisfies } \text{Task}(q_0, q_1) \wedge \zeta_s[i : f] \text{ satisfies } \text{Task}(q_1, q_2), \dots, \text{Task}(q_{f-1}, q_f)] \geq \eta \quad (2)$$

We also know that:

$$\Pr_{\zeta \in \mathcal{Z}}[\zeta \text{ satisfies } \mathcal{G}_\phi] := \Pr_{\zeta \in \mathcal{Z}}[\zeta_s[0 : i] \text{ satisfies } \text{Task}(q_0, q_1) \wedge \zeta_s[i : k] \text{ satisfies } \text{Task}(q_1, q_2), \dots, \text{Task}(q_{f-1}, q_f)] \quad (3)$$

Combining the above two equations:

$$\Pr_{\zeta \in \mathcal{Z}}[\zeta \text{ satisfies } \mathcal{G}_\phi] \geq \eta$$

Thus,

$$\Pr_{\zeta \in \mathcal{Z}}[\zeta \text{ satisfies } \phi] \geq \eta$$

Algorithm 1: *LSTS* ($\mathcal{G}_\phi, M, \eta, x$)

Output: Set of learned policies : Π^* , Edge-Policy Dictionary \mathcal{P}

```

1: Placeholder Initialization:
2: Set of Active Tasks:  $AT \leftarrow \emptyset$ ; Set of Learned Tasks:  $LT \leftarrow \emptyset$ ; Set of Discarded Tasks:  $DT \leftarrow \emptyset$ 
3: Edge-Policy Dictionary  $\mathcal{P} : e \rightarrow \pi$  {Mapping an edge (a task)  $e$  to a randomly initialized policy}
4: Teacher Q-Value Dictionary:  $Q : e \rightarrow -\infty$  {Initialize all Teacher Q-Values to  $-\infty$ }
5: Algorithm:
6:  $\mathcal{X} \leftarrow \text{Adjacency\_Matrix}(\mathcal{G}_\phi)$  {Get the adjacency matrix from the DAG representation}
7:  $AT \leftarrow AT \cup \{\mathcal{X}[q_0]\}$  {Add all outgoing edges from  $q_0$  to  $AT$ }
8:  $Q[e] = 0 \forall e \in AT$  {Initialize Q-values to 0 for all tasks from the initial node  $q_0$ }
9: while True do
10:    $e \leftarrow \text{Sample}(Q, AT)$  {Sample an edge (a task) from  $AT$  according to the Teacher Q-Values}
11:    $\bar{g} = []$ 
12:   for  $y \in x$  do
13:     task_outcome,  $\mathcal{P}[e], g \leftarrow \text{Learn\_Episode}(M, \mathcal{G}_\phi, e, \mathcal{P})$  {Learn task  $e$  for 1 episode}
14:      $\bar{g}.append(g)$ 
15:      $\bar{E}_{AT} \leftarrow \text{Next\_Tasks}(\mathcal{X}, e, DT)$ 
16:     if task_outcome = Success and  $|\bar{E}_{AT}| \neq 0$  then
17:       while True do
18:          $\tilde{e} \leftarrow \text{Sample}(Q, \bar{E}_{AT})$  {Sample a task from  $\bar{E}_{AT}$ }
19:         task_outcome,  $\mathcal{P}[\tilde{e}], g \leftarrow \text{Learn\_Episode}(M, \mathcal{G}_\phi, \tilde{e}, \mathcal{P})$ 
20:         Update_Teacher( $Q, \tilde{e}, g$ ) {Update the Teacher Q-Values}
21:          $\bar{E}_{AT} \leftarrow \text{Next\_Tasks}(\mathcal{X}, \tilde{e}, DT)$ 
22:         if task_outcome = Success and  $|\bar{E}_{AT}| \neq 0$  then
23:           continue
24:         else
25:           break
26:         end if
27:       end while
28:     end if
29:   end for
30:   Update_Teacher( $Q, e, \text{mean}(\bar{g})$ ) {Update the Teacher Q-Values}
31:   if Convergence( $Q, e, g, \eta$ ) then
32:      $\Pi^* \leftarrow \Pi^* \cup \mathcal{P}[e]$ ;  $LT \leftarrow LT \cup \{e\}$ ;  $AT \leftarrow AT \setminus \{e\}$  {Update the sets. ' $\setminus$ '  $\rightarrow$  set-minus}
33:      $Q[e] = -\infty$  {Set Q-Value for the learned task to  $-\infty$  so it is not sampled anymore}
34:      $\bar{E}_{DT} \leftarrow \text{Discarded\_Tasks}(\mathcal{X}, e)$  {Find any tasks that need to be discarded}
35:      $DT \leftarrow DT \cup \bar{E}_{DT}$  {Add the identified tasks that need to be discarded to  $DT$ }
36:      $\forall \bar{e} \in \bar{E}_{DT} : Q[\bar{e}] = -\infty$  {Set Teacher Q-Value for the discarded tasks to  $-\infty$ }
37:      $\bar{E}_{AT} \leftarrow \text{Next\_Tasks}(\mathcal{X}, e, DT)$  {Find new tasks that can be sampled}
38:     if  $|\bar{E}_{AT}| = 0$  then
39:       break {break once we reach  $q \in F$ }
40:     end if
41:      $\forall \bar{e} \in \bar{E}_{AT} : Q[\bar{e}] = 0$  {Set Teacher Q-Value for new tasks = 0 so they can be sampled}
42:      $AT \leftarrow AT \cup \bar{E}_{AT}$  {Add the new tasks to the set of Active Tasks  $AT$ }
43:   end if
44: end while
45: return  $\Pi^*, \mathcal{P}$ 

```

C Baselines

In this section, we present details on how the baseline approaches used in the paper are implemented.

C.1 DiRL and DiRL^c

DiRL (Jothimurugan et al. 2021) interleaves a high-level planning algorithm to guide the agent to learn those policies on which the agent shows the highest learning rate. The aim is to start from the initial node of the graphical representation of the high-level objective, and learning policies for all tasks (edges) from the initial node. Once the agent spends K interactions on each of the edge, the approach finds out which policy yields the highest success rate, and then the agent attempts to learn the policies from all edges from the current node. This process goes on iteratively, where the algorithm uses Dijkstra’s algorithm to find out paths to a node that have the highest success rate, and the agent explores all edges from that node. This process goes on iteratively until the agent reaches the final node. In this approach, the value for K needs to be manually determined, which depends on having an estimate of how many interactions are needed to learn the task. For our approach, we manually set this value to 20000 episodes of 100 interactions each. This is because for certain seeds, the task did not show any success if the value for K was less than 20000. The policy for the edge $q_0 \xrightarrow{Key_2} q_2$ requires 20000 episodes of 100 interactions to achieve a successful policy of 95% success rate. While DiRL experiments with a number of different values for K and plots for the mean, we could have increased the value for K but that would have only increased the number of interactions needed to get a successful policy for the final task.

We also implemented a custom implementation of DiRL, which we call DiRL^c, where we did not explore the task any further once the agent achieved a mean success rate of 95% and if the policy did not improve any further on that task. This helped the agent save costly interactions on the tasks where a successful policy was achievable in much fewer than 20000 episodes (tasks corresponding to the edges $q_2 \xrightarrow{Door} q_3$, $q_2 \xrightarrow{Door} q_3$ and $q_3 \xrightarrow{Goal} q_4$). However, the agent did spend all those interactions on the unpromising task corresponding to the edge $q_1 \xrightarrow{Door} q_3$, and our proposed *LSTS* was able to move ahead in the graph without having to spend all those interactions on the unpromising task. We call this approach DiRL^c where ‘c’ refers to converged.

C.2 Teacher-Student Curriculum Learning

The Teacher-Student Curriculum Learning approach (Matisen et al. 2020) assumes a Teacher agent optimizing the sequence of the tasks in the curriculum for the Student. The Teacher proposes those tasks for which the Student shows the highest potential in learning, i.e. the tasks for which the Student learns the quickest. The ability of a Student to learn a task is given by its slope of the curve of plotted reward function. The steeper the slope, the quicker the Student is able to learn the task. One limitation of this approach is that it assumes the source tasks of the curriculum are defined.

This is difficult in settings where the parameters of the task are continuous. We used the same PPO network (Hyperparameters in Section E) for Teacher-Student curriculum learning approach as we did for our proposed approach *LSTS*. We initialized 3 source tasks and one target task, with varying parameters for each source task. The tasks were: (1) Collect *Key*₁; (2) Collect *Key*₂; (3) Open *Door*; (4) Final task of reaching *Goal*. For each task, the agent starts from an initial state of the environment, and attempts a task proposed by the Teacher agent. For a fair comparison, the number of *interactions allocated* for this task is 500 to acknowledge the difficulty of the entire task.

For the tabular results in the paper, the performance of the agent in the final target task is displayed. The agent was not able to learn a successful policy for the entire task.

In Teacher-Student curriculum learning, the sunk cost of the algorithm is the interactions in the source task. The sunk cost is very high when the interactions are costly, as for our robotic environments.

C.3 Q-Learning for Reward Machines

QRM (Icarte et al. 2018) decomposes the task into sub-goals and intends to learn one q-value function for each state in the automaton. As an example, consider the automaton shown in Fig.1(b), each of the nodes in the automaton will have a corresponding q-value function. At all times during the training, the agent keeps track of its state in the automaton. While choosing an action given the current observation, the agent chooses an action depending on its current state in the Reward Machine. For example, if the agent is in node q_1 , it will choose an action given by the q-value function q_{q_1} corresponding to the node q_1 . While updating the q-value function, we use the Reward Machine to determine the reward the agent would have received had it been in the reward state given by the action chosen, i.e. to update

$$q_q \leftarrow \alpha r(s, a, s') + \gamma \max_{a'} q_p(s', a')$$

Here, the maximization is over the q_p that determines the new reward state of the machine. For our implementation, all transitions in the Reward Machine achieve a reward of 0, and any transition that achieves a goal state $q \in F$ achieves a reward of 1. For a fair comparison, the number of *interactions allocated* for this task is 500 to acknowledge the difficulty of the entire task.

We implemented this approach for the grid-world domain with the SPECTRL objective:

$$(Key_1 | Key_2) \& X (Door \& X (Goal)) \quad (4)$$

C.4 Guiding Search via Reward Shaping

GSRS (Camacho et al. 2018) shapes the reward inversely proportional to the distance from the accepting node in the automaton. The reward shaping is given by: $R'(s, a, s') = R(s, a, s') + F(s, a, s')$ where R is the original reward function and F is the shaping function. Here, F is given by:

$$F(s, a, s') = \gamma q(s') - q(s)$$

Here, F does not depend on the previous 2 states, but the last two states visited in the Reward Machine. This ensures that

| Parameter | Value |
|------------------------------------|---|
| discount factor γ | 0.99 |
| learning rate α | 1×10^{-3} |
| Optimizer | Adam |
| PPO clipping parameter | 0.2 |
| GAE λ | 0.95 |
| Entropy regularization coefficient | 0.001 |
| Entropy regularization coefficient | 0.001 |
| Action distribution | Categorical with 6 bins |
| Network architecture | 2 Conv layers of [64, 64] followed by 2 linear layers with <i>relu</i> activation |

Table 1: Parameters used for training the Proximal policy optimization

a positive shaping reward is given to transitions that make progress toward the goal state in the automaton. We shape the reward incrementally from the start state in the automaton until the accepting state, such that everytime the agent takes an action that triggers a transition in the automaton to a different state, a positive reward is given that is higher than the previous reward. In the end, to preserve optimality, we subtract the shaped reward from the sum of rewards gained. For a fair comparison, the number of *interactions allocated* for this task is 500 to acknowledge the difficulty of the entire task. This approach tends to reach local optima as the states with Key_1 and Key_2 will have similar reward settings, however reaching Key_1 makes the problem harder to solve as the path from Key_1 to the *Door* requires a much longer trajectory.

D Statistical Significance

To demonstrate that the average convergence rate of *LSTS* is consistently higher than the baseline approaches, we perform an unpaired t-test (Kim 2015). For the experiment, we consider a confidence interval of 95% and evaluate the p-value between *LSTS* approach and the best-performing baseline approach by comparing the success rates of the two approaches on 100 episodes after training for 10^7 interactions. Thus, through the results, we see that our proposed approach, *LSTS* has a consistent performance in the convergence rate. The results are always statistically significant. In all the experiments, *LSTS* has a much more sample-efficient performance. Thus, *LSTS* not only achieves a better success rate, but also adapts to converges quicker.

E Hyperparameters

Table 1 summarizes the hyperparameters for the PPO (Schulman et al. 2017) used for the gridworld and the TurtleBot navigation experiments.

All the experiments were conducted using a 64-bit Linux Machine, having Intel(R) Core(TM) i9-9940X CPU @ 3.30GHz processor, 126GB RAM memory and an Nvidia 2080 GPU. The maximum duration for running the experiments was set at 24 hours.

For training the Panda arm environment, we used the de-

fault implementation of DDPG-HER from OpenAI baselines (Dhariwal et al. 2017; Gallouédec et al. 2021).

F Search-and-Rescue Task

In the search-and-rescue domain, the task of the agent is open a door using a key, then collect a fire extinguisher to extinguish the fire, and then find and rescue stranded survivors. The exact order in which the tasks are accomplished does not matter, however the environment is configured in such a way that certain sub-tasks are considerably harder for the RL agent. For example, in the path from the fire extinguisher to the fire, the agent will witness survivors. If the task of the agent is to extinguish fire, it will not rescue the survivors, and will learn policies that result in sub-optimal path. Hence, the correct path in the DAG will encourage the agent to rescue survivors initially before moving on to other aspects of the task objective.

The setup of the environment is similar to the Door-Key environment, with the agent acting in a grid of 14×14 . The agent has access to additional actions such as *extinguish* that extinguishes the *fire* object if the agent has picked up the *fire extinguisher* in its inventory. Additionally, the agent has access to *rescue* actions that rescues the survivors if the agent is near the survivors and facing them.

The aim of the experiment was to show that even with increasing task complexity, and bigger state and action spaces, our proposed approach, *LSTS* achieves sample-efficient learning as compared to the baseline approaches. Our approach is effective in finding unpromising tasks based on the learning progress of the agent and discard them. This helps the agent to attain successful policies for solving the task in fewer environmental interactions.

References

- Camacho, A.; Chen, O.; Sanner, S.; and McIlraith, S. A. 2018. Non-Markovian rewards expressed in LTL: Guiding search via reward shaping (extended version). In *GoalsRL, a workshop collocated with ICML/IJCAI/AAMAS*.
- Dhariwal, P.; Hesse, C.; Klimov, O.; Nichol, A.; Plappert, M.; Radford, A.; Schulman, J.; Sidor, S.; Wu, Y.; and Zhokhov, P. 2017. OpenAI Baselines. <https://github.com/openai/baselines>.
- Gallouédec, Q.; Cazin, N.; Dellandréa, E.; and Chen, L. 2021. panda-gym: Open-Source Goal-Conditioned Environments for Robotic Learning. *4th Robot Learning Workshop: Self-Supervised and Lifelong Learning at NeurIPS*.
- Icarte, R. T.; Klassen, T.; Valenzano, R.; and McIlraith, S. 2018. Using reward machines for high-level task specification and decomposition in reinforcement learning. In *Intl. Conf. on Machine Learning*, 2107–2116.
- Jothimurugan, K.; Bansal, S.; Bastani, O.; and Alur, R. 2021. Compositional reinforcement learning from logical specifications. *Neural Information Processing Systems*.
- Kim, T. K. 2015. T test as a parametric statistic. *Korean journal of anesthesiology*, 68(6): 540–546.
- Matiisen, T.; Oliver, A.; Cohen, T.; and Schulman, J. 2020. Teacher-Student Curriculum Learning. *IEEE Trans. Neural Networks Learn. Syst.*, 31(9): 3732–3740.

Schulman, J.; Wolski, F.; Dhariwal, P.; Radford, A.; and Klimov, O. 2017. Proximal Policy Optimization Algorithms. *CoRR*.